88 再次重温写出各种SQL语句的时候，会用什么执行计划？（1）

今天开始，我们将用连续三篇文章给大家去重温平时我们写的SQL语句在执行的时候会用什么样的执行计划，因为我们讲完了SQL语句使用索引的规则和规律，也讲过了不同的使用索引的方法对应着执行计划里的什么访问方式，接下来就可以重温一下，直接把我们平时写的SQL语句和执行计划关联起来了。

首先，我们已经学习了const、ref、range、index几种执行计划里的访问方式，const、ref和range本质都是基于索引查询，只要你索引查出来的数据量不是特别大，一般性能都极为高效，index稍微次一点，需要遍历某个二级索引，但是因为二级索引比较小，所以遍历性能也还可以的。

另外最次的一种就是all了，all意思就是直接全表扫描，扫描你的聚簇索引的所有叶子节点，也就是一个表里一行一行数据去扫描，如果一个表就几百条数据那还好，如果是有几万条，或者几十万，几百万数据，全表扫描基本就得跪了。

那么大家对之前讲的一些特别简单的SQL语句，其实都知道会用什么样的执行计划和访问方式了，也知道不同的访问方式是如何使用索引的，今天开始我们来继续讲讲更多的SQL语句你写出来之后，会用什么样的执行计划。

首先大家看一个SQL语句：select * from table where x1=xx or x2>=xx，这个SQL语句要查一个表，用了x1和x2两个字段，此时有人可能会说了，要是你对x1和x2建了一个联合索引，那不就直接可以通过索引去扫描了？

但是万一要是你建的索引是两个呢？比如(x1,x3)，(x2,x4)，你建了两个联合索引，此时你这个SQL只能选择其中一个索引去用，此时会选择哪个呢？这里MySQL负责生成执行计划的查询优化器，一般会选择在索引里扫描行数比较少的那个条件。

比如说x1=xx，在索引里只要做等值比较，扫描数据比较少，那么可能就会挑选x1的索引，做一个索引树的查找，在执行计划里，其实就是一个ref的方式，找到几条数据之后，接着做一个回表，回到聚簇索引里去查出每条数据完整数据，接着加载到内存里，根据每条数据的x2字段的值，根据x2>=xx条件做一个筛选。

这就是面对两个字段都能用索引的时候如何选择，以及如何进行处理的方式。

接着我们再来考虑另外一种情况，就是：select * from table where x1=xx and c1=xx and c2>=xx and c3 IS NOT NULL

其实我们平时经常会写出来类似这样的SQL语句，就是在一个SQL的所有筛选条件里，就一个x1是有索引的，其他字段都是没有索引的。

这种情况其实也是非常常见的，一般我们在写好一个系统之后，针对所有的SQL分析时，当然不可能针对所有的SQL里的每一个where里的字段都加一个索引，那是不现实的，最终我们只能在所有的SQL语句里，抽取部分经常在where里用到的字段来设计两三个联合索引。

所以在这种情况下，必然很多SQL语句里，可能where后的条件有好几个，结果就一个字段可以用到索引的，此时查询优化器生成的执行计划，就会仅仅针对x1字段走一个ref访问，直接通过x1字段的索引树快速查找到指定的一波数据。

接着对这波数据都回表到聚簇索引里去，把每条数据完整的字段都查出来，然后都加载到内存里去。接着就可以针对这波数据的c1、c2、c3字段按照条件进行筛选和过滤，最后 拿到的就是符合条件的数据了。

所以你的x1索引的设计，必然尽可能是要让x1=xx这个条件在索引树里查找出来的数据量比较少，才能保证后续的性能比较高。

End